WE CLAIM:

1. A method for preparing executables for execution in a distributed processing environment in accordance with a set of process representations of business logic, the method comprising the steps of:

providing process representations in a process calculus notation;

verifying that the representations are valid;

generating executables and corresponding test data in accordance with the verified representations; and

testing the executables using the corresponding test data.

2. A method according to claim 1, further comprising the step of deploying the tested executables in the distributed processing environment.

3. A method according to claim 2, further comprising the step of monitoring the performance of the deployed executables to gather process execution information.

4. A method according to claim 3, further comprising the steps of analysing information gathered in the monitoring step; and autonomically altering the executables and corresponding test data in accordance with analysed process execution information.

5. A method according to claim 4, wherein the step of autonomically altering the executables includes altering the generation of executables and test data in accordance with analysed process execution information.

6. A method according to claim 4, wherein the step of autonomically altering the executables comprises altering the process representations and repeating the verification, generation and testing steps.

7. A method according to claim 4, wherein the step of autonomically altering the executables is furthermore performed in accordance with contextual information.

8. A method according to claim 7, wherein the contextual information includes heuristics.

9. A method according to claim 1, wherein the step of generating the executables is furthermore performed in accordance with contextual information.

10. A method according to claim 4, wherein the step of autonomically altering the executables comprises altering the executables and test data directly and repeating the testing step.

11. A method according to claim 4, wherein the step of autonomically altering the executables comprises comparing the analysed process execution information with an earlier set of process representations and altering the executables to reduce significant disparities between them.

12. A method according to claim 11, further comprising the step of repeating the generating, testing, analysing and altering steps until the comparison indicates the absence of significant disparity.

13. A method according to claim 1, wherein the executables are generated as source code in a third generation language.

14. A method according to claim 13, wherein the third generation language is one of the set of languages including C, C++, C#, Ada, Java, Delphi, Visual Basic, and FORTRAN 90.

15. A method according to claim 1, wherein the process calculus notation is based upon XML.

16. A method according to claim 13, wherein the process calculus notation is RIFML.

17. A computer program product for preparing executables for execution in a distributed processing environment in accordance with a set of process representations of business logic, the product comprising:

a datastore for storing process representations in a process calculus notation;

a verification module, for verifying that the representations are valid;

a generator module for generating executables and corresponding test data in accordance with the verified representations; and,

a tester module for testing the executables using the corresponding test data.

18. A computer program product according to claim 17, wherein the tester module allows the tested executables to be deployed in the distributed processing environment.

19. A computer program product according to claim 18, further comprising a monitor module, for monitoring the performance of the deployed executables to gather process execution information.

20. A computer program product according to claim 19, further comprising an analyser module for analysing process execution information gathered by the monitor module and an update module, which alters the executables and corresponding test data autonomically in accordance with analysed process execution information.

21. A computer program according to claim 20, wherein the autonomic alteration of the executables and corresponding test data includes alteration of the behaviour of the generator module in accordance with analysed process execution information.

22. A computer program product according to claims 20, wherein the update module autonomically alters the executables and corresponding test data by altering the process representations and instigating the sequential operation of the verification module, the generator module, the tester module and the analyser module.

23. A computer program product according to claims 20, wherein the update module autonomically alters the executables in accordance with contextual information.

24. A computer program product according to claim 23, wherein the contextual information includes heuristics.

25. A computer program product according to claim 15, wherein generator module generates the executables in accordance with contextual information.

26. A computer program product according to claim 20, wherein the update module autonomically alters the executables and corresponding test data by altering the executables and test data directly and instigating the operation of the tester module and the analyser module.

27. A computer program product according to claim 20, wherein the update module autonomically alters the executables and corresponding test data by comparing the analysed process execution information with the earlier set of process representations and altering the executables to reduce significant disparities between them.

28. A computer program product according to claim 20, wherein the sequential operation of the verification module, the generator module, the tester module, the analyser module and the update module continues autonomically until the comparison indicates the absence of significant disparity.

29. A computer program product according to claim 17, wherein the executables are generated as source code in a third generation language.

30. A computer program product according to claim 29, wherein the third generation language is one of the set of languages including C, C++, C#, Ada, Java, Delphi, Visual Basic, and FORTRAN 90.

31. A computer program product according to claim 17, wherein the process calculus notation is based upon XML.

32. A method according to claim 31, wherein the process calculus notation is RIFML.

33. A computer system for preparing executables for execution in a distributed processing environment in accordance with a set of process representations of business logic, the system comprising:

a datastore for storing process representations in a process calculus notation;

a verification module, for verifying that the representations are valid;

a generator module for generating executables and corresponding test data in accordance with the verified representations; and

a tester module for testing the executables using the corresponding test data.

34. A computer system according to claim 33, wherein the tester module allows the tested executables to be deployed in the distributed processing environment.

35. A computer system product according to claim 18, further comprising a monitor module, for monitoring the performance of the deployed executables to gather process execution information.

36. A computer system product according to claim 19, further comprising an analyser module for analysing process execution information gathered by the monitor module and an update module, which alters the executables and corresponding test data autonomically in accordance with analysed process execution information.

37. A computer system according to claim 20, wherein the autonomic alteration of the executables and corresponding test data includes alteration of the behaviour of the generator module in accordance with analysed process execution information.

38. A computer system according to claim 36, wherein the update module autonomically alters the executables and corresponding test data by altering the process representations and instigating the sequential operation of the verification module, the generator module, the tester module and the analyser module.

39. A computer system according to claims 36, wherein the update module autonomically alters the executables in accordance with contextual information.

40. A computer system according to claim 39, wherein the contextual information includes heuristics.

41. A computer program product according to claim 33, wherein generator module generates the executables in accordance with contextual information.

42. A computer system according to claim 36, wherein the update module autonomically alters the executables and corresponding test data by altering the executables and test data directly and instigating the operation of the tester module and the analyser module.

43. A computer system according to claim 36, wherein the update module autonomically alters the executables and corresponding test data by comparing the analysed process execution information with the earlier set of process representations and altering the executables to reduce significant disparities between them.

44. A computer system according to claim 43, wherein the sequential operation of the verification module, the generator module, the tester module, the analyser module and the update module continues autonomically until the comparison indicates the absence of significant disparity.

45. A computer system according to claim 33, wherein the executables are generated as source code in a third generation language.

46. A computer program product according to claim 45, wherein the third generation

language is one of the set of languages including C, C++, C#, Ada, Java, Delphi, Visual Basic, and FORTRAN 90.

47. A computer program product according to claim 33, wherein the process calculus notation is based upon XML.

48. A method according to claim 47, wherein the process calculus notation is RIFML.

49. A method for verifying and implementing process representations of business logic for execution in a distributed processing environment, the method comprising:

providing process representations in a process calculus notation;

verifying that the representations are valid;

generating executables and corresponding test data in accordance with the verified representations;

testing the executables using the corresponding test data;

analysing process execution information; and

autonomically altering the executables and corresponding test data in accordance with said analysed process execution information.

50. A computer program product for verifying and implementing process representations of a business logic in a distributed processing environment, the product comprising:

means for storing process representations in a process calculus notation;

means for verifying that the representations are valid;

means for generating executables and corresponding test data in accordance with the verified representations;

means for testing the executables using the corresponding test data;

means for analysing process execution information; and

means for altering the executables and corresponding test data autonomically in accordance with analysed process execution information.

51. A computer system for verifying and implementing process representations of business logic in a distributed processing environment, the system comprising:

means for storing process representations in a process calculus notation;

means for verifying that the representations are valid;

means for generating executables and corresponding test data in accordance with the verified representations;

means for testing the executables using the corresponding test data;

means for analysing process execution information; and

means for altering the executables and corresponding test data autonomically in accordance with analysed process execution information.